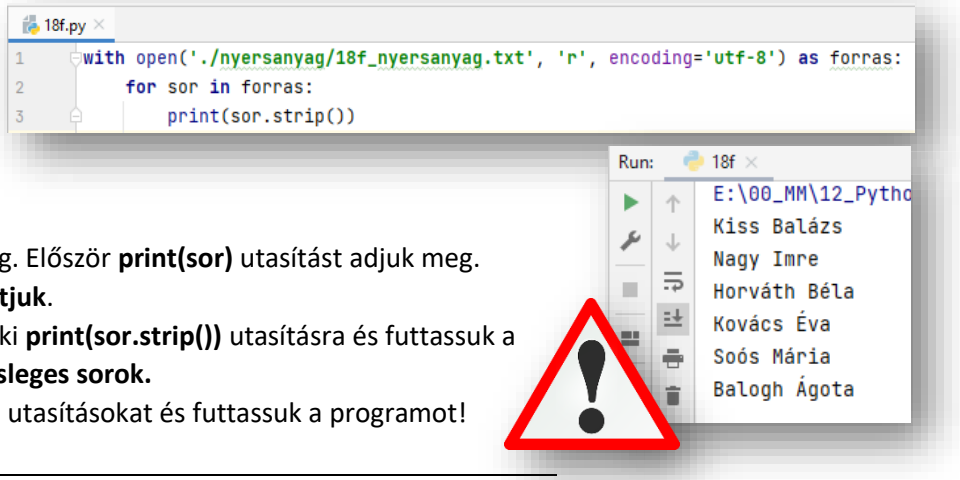
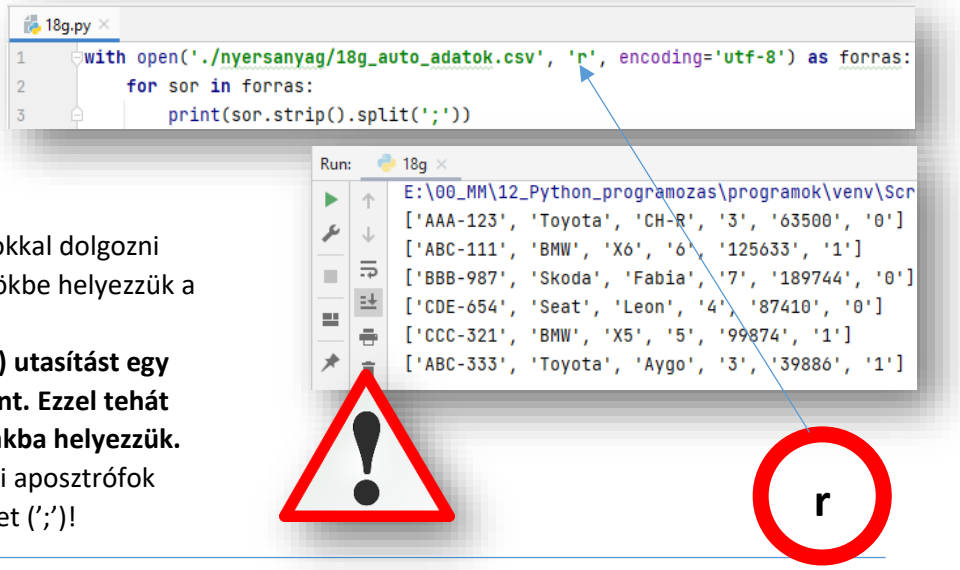
(18f.py)



Tehát a memória túlcsordulás kivédésére soronként olvassuk be a 18f_nyersanyag.txt szövegfájl karaktereit.
Ezt a for utasítással tesszük meg. Először **print(sor)** utasítást adjuk meg.
Ekkor még az **üres sorokat is látjuk**.
Ennek „kivédésére” egészítsük ki **print(sor.strip())** utasításra és futtassuk a programunkat. **Eltűnnek a felesleges sorok.**
A minta alapján értelmezzük az utasításokat és futtassuk a programot!

(18g.py)

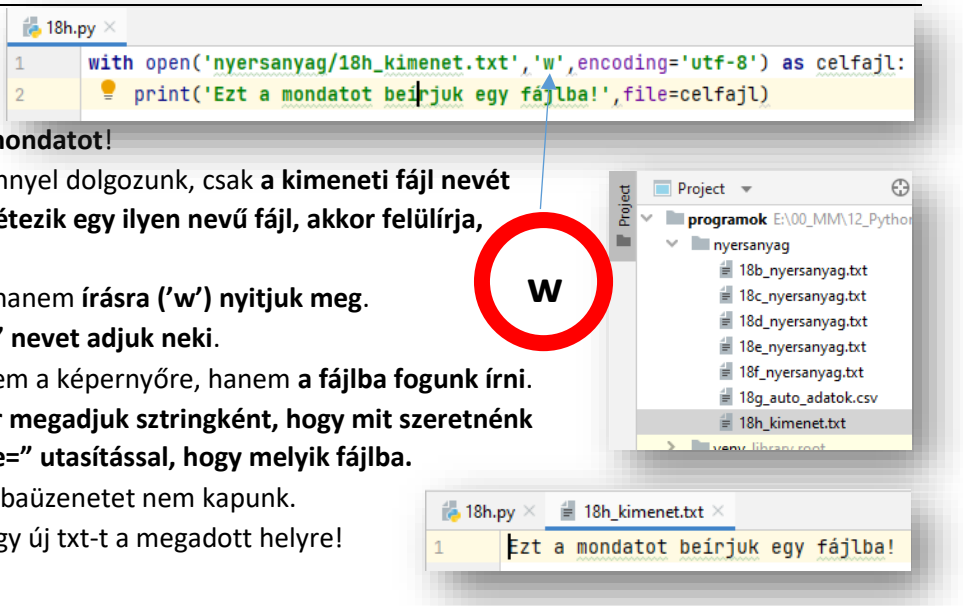


A txt fájlokon kívül **csv fájlokkal is tudunk dolgozni**. Olvassuk be a 18g_auto_adatok.csv fájlt soronként.
Ahhoz, hogy a beolvasott adatokkal dolgozni tudjunk a későbbiekben, tömbökbe helyezzük a sztringeket.
Egészítsük ki a print(sor.strip()) utasítást egy split() utasítással a minta szerint. Ezzel tehát feldaraboljuk sorainkat és listákba helyezzük.
A zárójelek között meg kell adni aposztrófok között az elválasztó karaktereket (’;’)!

Adatok kiírása fájlba, adatok módosítása

Az előzőekben megismerkedhettünk azzal, hogy hogyan tudunk fájlból adatokat beolvasni. Most pedig megnézzük, hogy hogyan tudunk a programunkból fájlba írni, menteni és így ezek az adatok képesek megmaradni a programokból való kilépés után is.

(18h.py)

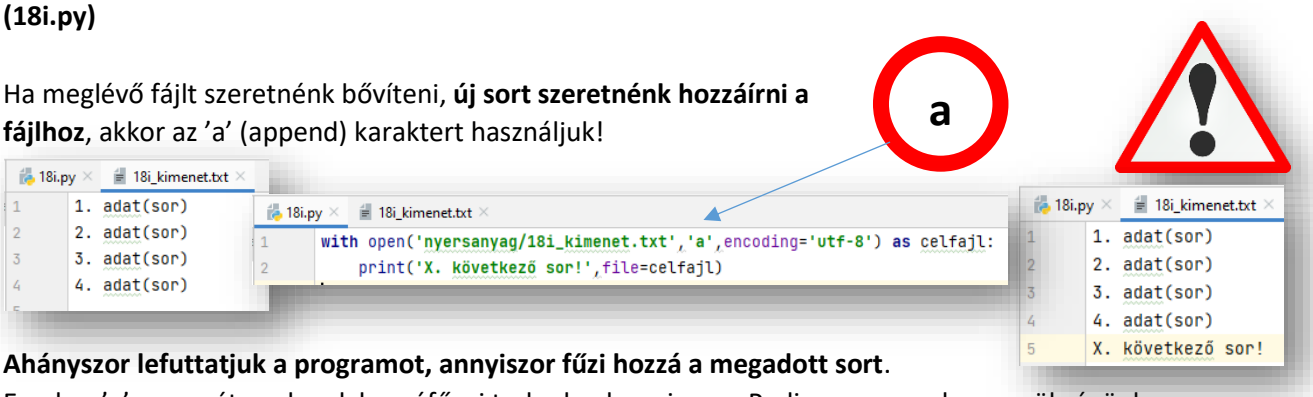


A következő programban egy **txt fájlba fogunk kiíratni egy mondatot**!

- Ugyanúgy az open függvénnyel dolgozunk, csak **a kimeneti fájl nevét és helyét adjuk meg. Ha létezik egy ilyen nevű fájl, akkor felülírja, ha nem akkor létrehozza!**
- Most nem olvasásra (’r’), hanem **írásra (’w’) nyitjuk meg**.
- Ebben az esetben **„celfajl” nevet adjuk neki**.
- A print utasítással most nem a képernyőre, hanem **a fájlba fogunk írni**.
- **A zárójelek között először megadjuk sztringként, hogy mit szeretnénk a fájlba íratni, majd a „file=” utasítással, hogy melyik fájlba.**
- Futtassuk a programot! Hibaüzenetet nem kapunk.
- Látjuk, hogy létrehozott egy új txt-t a megadott helyre!

(18i.py)

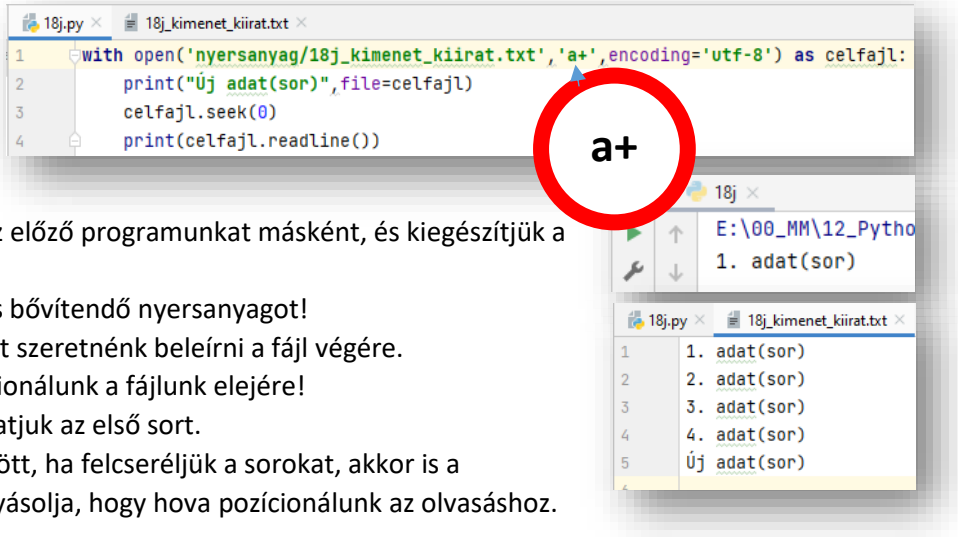
Ha meglévő fájlt szeretnénk bővíteni, **új sort szeretnénk hozzáírni a fájlhoz**, akkor az ’a’ (append) karaktert használjuk!



Ahányszor lefuttatjuk a programot, annyiszor fűzi hozzá a megadott sort.
Ezzel az ’a’ paraméterrel csak hozzáfűzni tudunk, olvasni nem. Pedig nagyon sokszor szükségünk van egyszerre a kettőre. Ezért a következő programban erre nézünk példát.

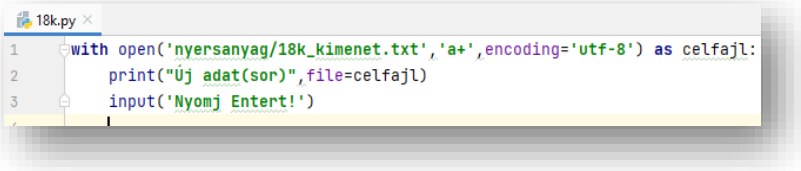
(18j.py)

Ha szeretnénk **fájlba íratni és képernyőre is íratni**,
akkor az ’a+’ paramétert kell alkalmaznunk.
Ebben az esetben elmentjük az előző programunkat másként, és kiegészítjük a mintán látható utasításokkal.
Tehát átírjuk a megnyitandó és bővítendő nyersanyagot!
Megadjuk a 2. sorban, hogy mit szeretnénk beleírni a fájl végére.
A harmadik sorban visszapozícionálunk a fájlunk elejére!
Majd readline() utasítással kiíratjuk az első sort.
A fájlmutató az olvasáshoz kötött, ha felcseréljük a sorokat, akkor is a fájl végére ír. Tehát nem befolyásolja, hogy hova pozícionálunk az olvasáshoz.

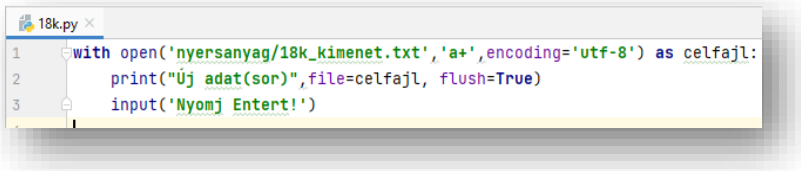


(18k.py)

Nézzük meg, hogy a **fájlba való kiíratás mikor történik** meg! A lefutás során, amikor a kiíratás sorához érünk, vagy a program lefutásának végén?

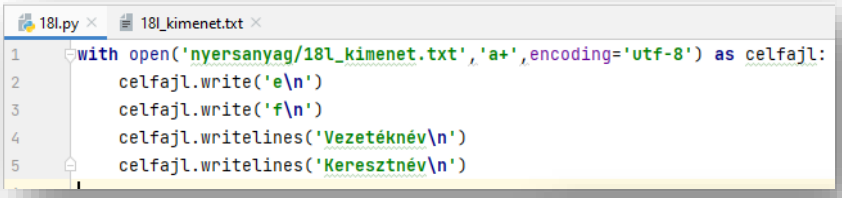


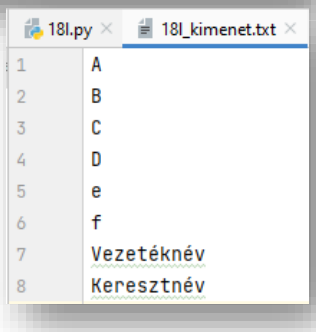
- Most fontos a műveletek sorrendje, úgyhogy haladjunk lépésről lépésre!
- Először gépeljük be a jobb oldalon lévő kódot!
- Az input() utasítással egy karakter lenyomására vár!
- Nyissuk meg a feladathoz tartozó nyersanyagot és a feladat lefutása során folyamatosan vizsgáljuk!
- Futtassuk a programot!
- Vár az enter lenyomására, a txt-ben még nem történt semmi.
- Amikor megnyomjuk az entert akkor kerül be az új sor a kimeneti fájl végére!
- Tehát kiderült, hogy nem azonnal menti a változásokat, hanem egy pufferben tárolja mindaddig, amig nem végzünk a program futásával.
- Ha azt szeretnénk, hogy azonnal legyen mentve, akkor egy újabb paramétert kell megadnunk a második sorunkban. Ez pedig a flush=’True’ paraméter.
- Ha így futtatjuk a programunkat, akkor amikor a 2. sor lefut azonnal bekerül a txt-be az új sor.



(18l.py)

Egy **másik példa a fájlba való íratásra**!
Használjuk write() és writelines() utasításokat a minta szerint!

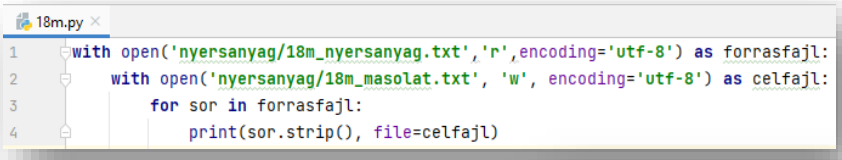


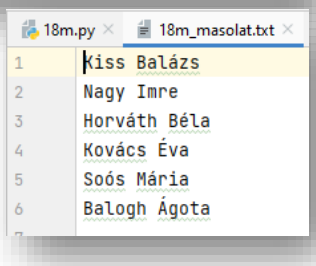


- Nyissuk meg a 18l_kimenet.txt fájlt írásra és olvasásra!
- Először próbáljuk ki a kódunkat „\n”-ek nélkül! Mi történik?
- Egymás mellé kiírja a karaktereket és a szavakat.
- Viszont ha használjuk a „\n” sortörést, akkor a képernyőképpel megegyező eredményt kapunk.

(18m.py)

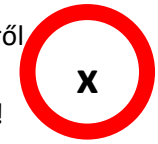
Nézzük meg, hogy **hogyan tudunk másolatot készíteni egy txt fájlról**!





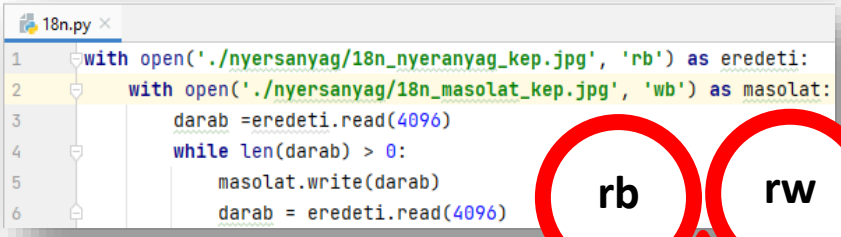
- A program első sorában megnyitunk olvasásra egy forrásfájlt!
- A második sorban megnyitunk írásra (új névvel) egy célfájlt!
- Soronként végig megyünk for utasítással a fájlunkon.
- A negyedik sorban a célfájlba tesszük – plusz enterek nélkül – az adatokat.
- Ellenőrizzük, hogy elkészült-e a másolat!

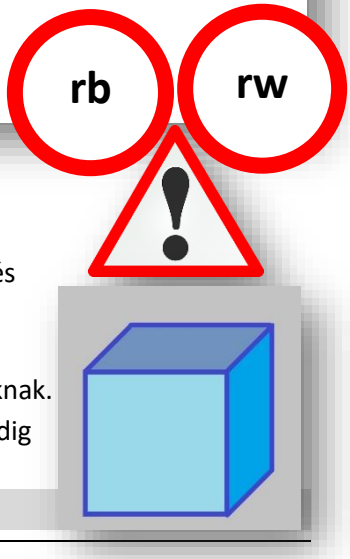
Eddig beszéltünk az r(olvasás) / w(írás) / a(végére írás) / a+(végére írás + olvasás) paraméterekről.
Ha az ’x’ paramétert használjuk, akkor ha már létezik a fájl, akkor nem engedi felülírni!
Ha nem létezik, akkor létrehozza, ha létezik, akkor hibaüzenetet küld! Kivételkezelés szükséges!



(18n.py)

Másoljunk le egy bináris fájlt! Mondjuk egy jpg képet!





- Ebben az esetben megnyitjuk az eredeti képet olvasásra – az első sorban - , de ebben az esetben, a bináris megnyitás miatt **’rb’ paramétert adunk meg!**
- Aztán a másodiksorban megadjuk, hogy a másolatot milyen néven mentsük, és itt ’wb’ paramétert adunk meg!
- A túlcsordulás miatt darabonként dolgozzuk fel (másoljuk) az adatokat, ezért a harmadik sorban, egy darab változóba beolvassuk egy kisebb részét a fájlinknak.
- Aztán mindaddig amig be tudunk olvasni a fájlból biteke (amíg nem nulla), addig másolatot készítünk a másolat nevű fájlba.
- Futtassuk a programunkat, ellenőrizzük a másolatot!